image processing framework, registration, fuzzy segmentation methods, visualization, MR brain image processing

Marcin DENKOWSKI[*], Michał CHLEBIEJ,
Paweł MIKOŁAJCZAK

REGISTRATION AND FUZZY SEGMENTATION MODULES FOR *SEMIVIS* FRAMEWORK

This paper presents the cross-platform framework for image processing with a focus on medical imaging. It allows a fast addition and testing of new algorithms using a modular structure. New modules can be created by using a platform-independent The C++ class library can be easily integrated with a whole system by a plug-in mechanisms. Together with the system core in the framework medical image processing modules are included. The plug-in mechanism allows to create a processing pipelines of this modules to achieve sophisticated processing functions such as registration or segmentation.

1. INTRODUCTION

Our image processing framework *SemiVis* was developed as a general platform for a medical pipeline, including data import, image processing and visualization. The main features of this system are its portability and ease of extension. Platform independence is achieved by using freely available cross-platform libraries like *Qt Toolkit* for user interface development and file handling [9] and *Kitware Visualization Toolkit (Vtk)* for image processing and rendering [8]. Extensibility is achieved by a plug-in mechanism. Developers can add functionality to *SemiVis* by implementing their own plug-ins and registering them with the framework. This can be done without recompilation of the source code. Since all core and plug-in classes are implemented in C++ the code can be used to generate executable programs on many systems. Together with the system core in the framework are included modules for medical image processing for such purposes as:

1. data managing;
2. 2D and 3D visualization with interaction tools;
3. image processing [4] (filtering, segmentation, analysis, 2D editing);
4. registration;
5. image and movie generation for demonstration purposes.

The whole system is intended to be open source software published under a GPL license [7]. For that reason all files and structures formats used by this system must be open standards (for example XML [10], DICOM [6]).

[*] Laboratory of Information Technology, Institute of Physics, Maria Curie Skłodowska University,
Pl. M. Curie-Skłodowskiej 1, 20-031 Lublin, Poland

2. *SEMIVIS* FRAMEWORK

Functionality of this framework is divided between two main parts: system core and modules. Even though structure of this system is complicated its usage must be intuitive and conventional. Therefore both main parts, the core and modules, must not have too many use cases that interact with user, or existing complicated use cases must be partitioned to form a hierarchical structure of simpler use cases [1]. Figure 1 presents the most general use case diagrams for two perspectives for both main parts of the system.

System Core perspective consists of three main use cases that are responsible for setting and saving application state as well as managing the application configuration. Module perspective involves four main use cases for managing modules, i.e. for adding and removing modules from the application, linking two modules and using modules. The last use case is extended by two other use cases: Update use case that releases module processing and Set Parameters use case for changing the module configuration. Of course every module has its own use case diagram that extends the shown general diagram, but that diagram depends on module purposes and its designer.

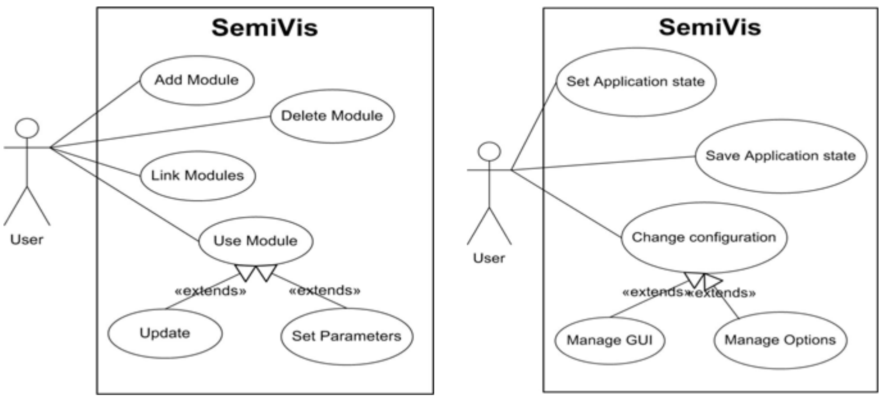


Fig. 1. Two use case perspectives for *SemiVis* framework

The whole framework is strongly module oriented and can not work without at least one module loaded. The interface between the system core and modules has been defined in a *cProcessObject* abstract class. This class offers operations common for all modules. Every module must derive from this class directly or indirectly. Every module class is also associated with a specific class derived from a *cProcessObjectWidget* class, which implements an interface for GUI representation of a module. The main principle of module function can be described as collecting data from its inputs, processing them and passing to its outputs. The module can have any number of entries and any number of exits. The modules can be connected into pipelines by connection exits form one module to the entry for the other module(s), but there are some limitations:

- one exit can propagate to any number of entries,
- one entry can be linked only to one exit.

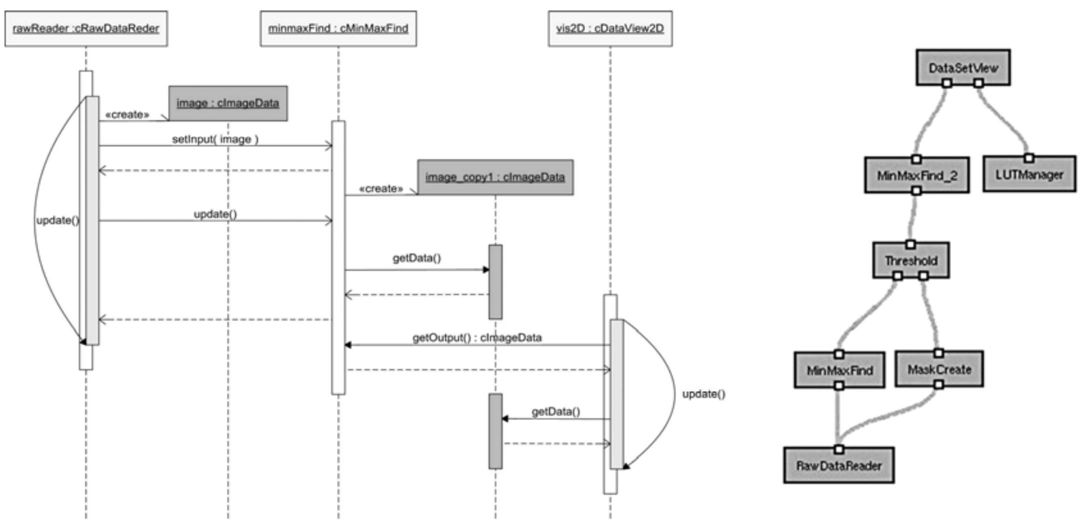
Data that are processed by modules are also encapsulated into classes derived from the common ancestor *cDataObject* class. The important feature of this class is the ability for sharing internal data between the objects of the same type. For that purposes a *reference count*, *deep copy* and *shallow copy* mechanisms are used. In the deep copy case a new copy object is an exact copy with all internal data of the original object and is completely independent. On the other hand for a shallow copy only a small part of internal data is copied but larger data parts are shared by both the original and copy objects.

Modules can be arranged into packages according to their specified functions:

1) input–output – modules for loading, saving, converting and maintaining various medical image standards (DICOM, TIFF, RAW [6]), other digital images (BMP, JPEG, PNG) and encapsulating these images into internal objects representation.
2) information, statistics – modules responsible for various informational and statistical outcome generation.
3) visualization – modules for graphical representation of dataset:
    1. 2D – allowing two-dimensional visualization of volumetric data from three orthogonal perspectives (transversal, sagittal and coronal), providing selection from a colour lookup-table (LUT) and supplying the masking mechanism for the dataset.
    2. 3D – providing a set of three-dimensional visualization tools like volume rendering (raycasting, texture mapping), surface rendering (isosurfaces rendered as triangle meshes) rendering together with a wide range of manipulation mechanisms providing a good spatial orientation in volumetric datasets. These modules rely heavily on OpenGL library [12].
    3. graph visualization – histogram of dataset, scatter-plot, and graphical representation of various data relationships.
4) segmentation – including modules for standard 3D image segmentation mechanisms (thresholding, region based, contour based, fuzzy logic).
5) filtering – providing a set of morphological, convolution and diffusion filters.
6) transformation – translation, rotation, rigid body, affine transformations.
7) registration – similarity measure, mutual information, optimization algorithm modules.

3. DATA PROCESSING PIPELINE

Every module is responsible for one, specific function performed on entry data. To achieve an aggregation of functions of several modules we can connect them into pipeline. Modules can be linked in either in series or in parallel. The aggregate action starts from the bottom module and propagates to upper modules as far as the current processed module is configured to be auto updated. An example action of 2-dimensional visualization of a data set is shown on the sequence diagram on Fig. 2a. The operator *User* starts the sequence by updating (*update()* operation) *rawReader* module. This module creates a data object *image («create»* signal*)*, loads data set from file and converts it into an internal format. At the next step the *rawReader* puts a properly prepared data object into the entry of the connected module and forces it to update (the *AutoUpdate* flag of this module is set to *true* value). This module creates its own shallow copy of the data object and processes this data. The third module in the shown pipeline is *vis2D* and has *AutoUpdate* flag set to *false* so that its update must be released by the user. In this case the module gets the data from its entry and processes it i.e. visualizes the data according to the module configuration. See Fig. 2b for a module processing pipeline using a graphical interface.



a) b)

Fig. 2. Example sequence diagram for data processing pipeline a), graphical UI representation of pipeline b)

4. GRAPHIC USER INTERFACE

The main factor that usually decides whether the application is functional or not is the way the program controls the data flow and communicates with the user. So the creation of an intelligible, flexible and useful graphical user interface is in principle a very important task. For this reason, the authors decided to use a multiple-document-interface (MDI) concept [5] with dockable windows from *Frames Package* and independent *ProcessObjectWindows* for each module. The central part of the application occupies the *Workspace* – the panel for defining data processing pipelines from module blocks. All application Frames can be docked to the *Workspace* or flattened around as independent windows. All modules have their own controlling main window designed independently for each module to fulfil their functionality. See Fig. 3 for the best illustration of graphical user interface.

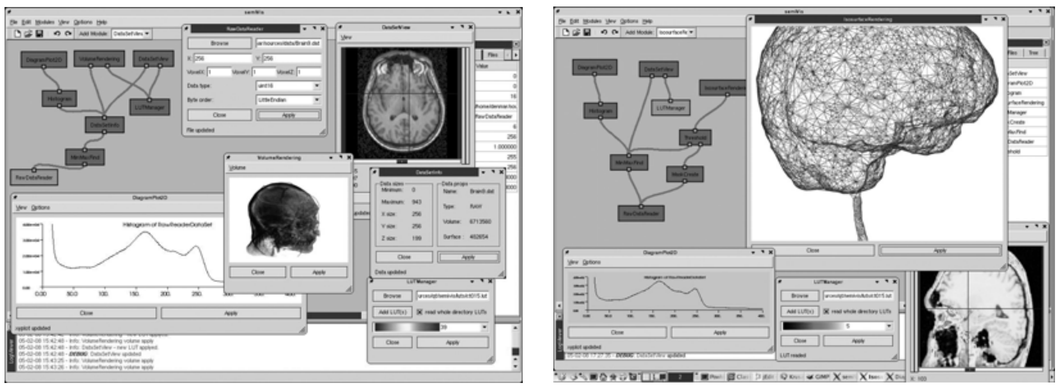


Fig. 3. GUI examples of *SemiVis framework* with simple visualization pipelines and windows of involved modules.

5. PHYSICAL STRUCTURE

The created system consists of several elements. The main one is the activating component *SemiVis* Core, which includes all the essential logical dependence mechanisms, GUI interface, and managing mechanisms. The modules making up the full functional system are delivered separately as dynamic shared system-dependent libraries (Linux - so, MS Windows – dll). These libraries need to harmonize with some internal components (i.e. OpenGL library). Configuration files are also included with the whole system package.

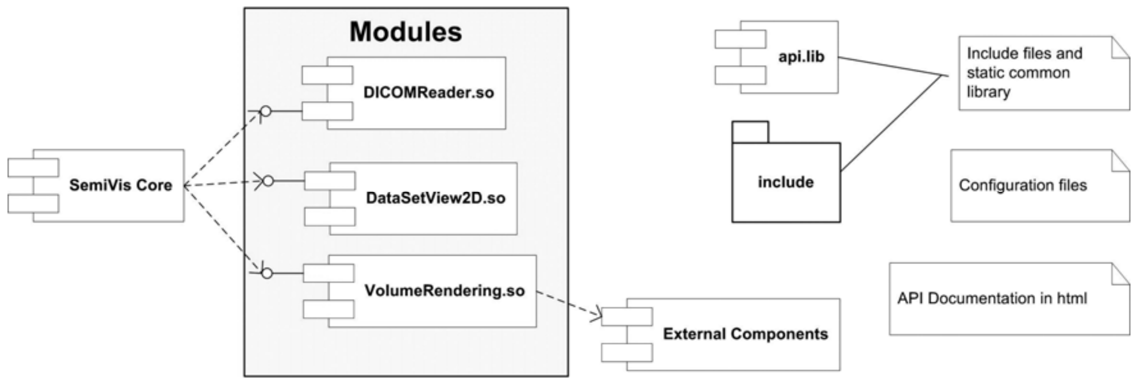


Fig. 4. *SemiVis* components diagram

An individual, and also very important, task is the problem of delivering the fully functional API interface allowing creation of personal components. The API consists of a static library including binaries and the header files package. The API documentation is also included as a set of html files. See Fig. 4 for the diagram of system components. The system requires at least a Pentium III processor. The lower limit of operating memory is 256 MB, but this may be insufficient for most medical datasets and long task pipelines. It is recommended installing at least 1 GB of RAM. For high quality and efficient visualization, the OpenGL compatible graphics card is indispensable. The application has been compiled and is currently being tested on a Linux system (kernel v. 2.4 +) and MS Windows (2000, XP).

6. REGISTRATION PIPELINE

The registration may be described as a process that aligns together two different objects. From the mathematical point of view the objective of the registration is to minimize a misregistration function between two datasets. To solve this problem using numerical methods we have to define a geometrical transformation relating the two data sets as well as a criterion for the goodness of a given transformation. We treat the registration problem as a non-linear optimization task aimed at the minimization of the dissimilarity measure. Given a pair of volumetric datasets (let us call them *model* and *object*) we can define the similarity measure as the objective function, which has to quantify how well both datasets match with each other. The transformation mapping T is defined as a geometric transformation that takes an *object* volume point and gives the corresponding *model* point. The majority of registration methods share a common optimization framework, where the goal is to estimate numerically the optimal transformation $T_{optimal}$, which establishes the optimal spatial correspondence between the *model* and the *object*. In the case of rigid body transformation (six degrees of freedom) a six-component vector $v=[rx\ ry\ rz\ tx\ ty\ tz]$ has to be estimated. In our system we have implemented two distinct methods based on different definitions of the similarity measure. The first one is the surface similarity measure.

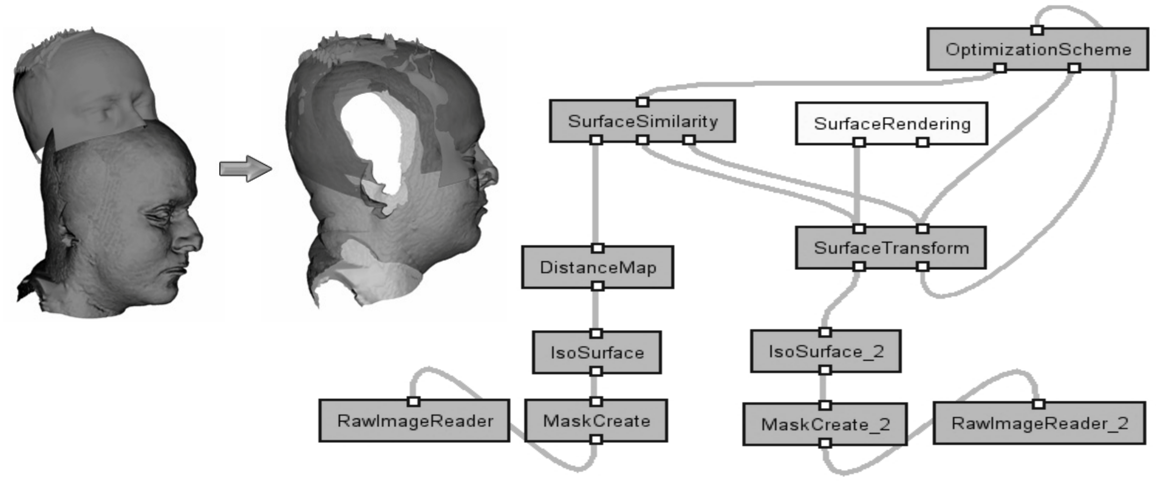


Fig. 5. Surface registration pipeline. At first iso-surfaces from two volumetric datasets are extracted. After that 3D distance map transform is calculated to provide real-time optimization process. At the main step optimization procedure (Levenberg-Marquardt) minimizes the surface similarity cost function for a given set of variable parameters. The optimization process is visualized using surface rendering technique.

The surface similarity approach belongs to non-automatic registration methods. The method requires the presence of two datasets defining the surfaces of both objects of interest. The datasets are binary volumes containing only values corresponding to the edges or boundaries of the objects in the original datasets. To obtain such surfaces a preprocessing stage is required. After this step, registration of *model* and *object* datasets can be performed.

Let $d(p)$ be Euclidean distance in $\Re^3$ between any transformed point p and the closest to p' data point p' and it denotes the length of the vector that is defined by this pair of points: $d(p) = \min\|p - p'\|$. For any vector $v = [r_x, r_y, r_z, t_x, t_y, t_z]$ in $\Re^3$ the surface similarity measure can be defined as follows:

$$C(v) = \sum_{i=1}^{N} d^2(T_v(p_i)) \qquad (1)$$

where $d^2(T_v(p_i))$ is the distance between the position of the *object* point p after being transformed by T and the closest point of the *model* as defined above. N is the number of points defining the extracted object surface. The method can be accelerated by computing a 3D Euclidean distance map dataset in the pre-processing phase. The distance transform is an operator that can be applied on the 3D binary surface dataset. Each single voxel in the created distance map is labelled with its distance from the nearest boundary point. Euclidean distance computation (in the opposite of chamfer, chessboard, and city-block distance transforms) allows calculation of all three spatial coordinates of a distance vector. These parameters can be used in the optimization process to estimate derivatives of the objective function. Because of that reason in the optimization process of the cost function we may apply methods following the gradient direction like Davidon-Fletcher-Powell [13] or Levenberg-Marquardt [11]. The whole process pipeline is presented in Figure 5.

A different approach of defining matching measure is to consider all voxels with corresponding grey-values in both datasets. Voxel based methods are more versatile and they do not need non-automatic pre-processing steps as surface similarity method does. There exists a very wide range of different voxel based registration methods. In comparison to surface similarity or landmark based registration techniques these methods are much more time consuming, which is their main drawback.

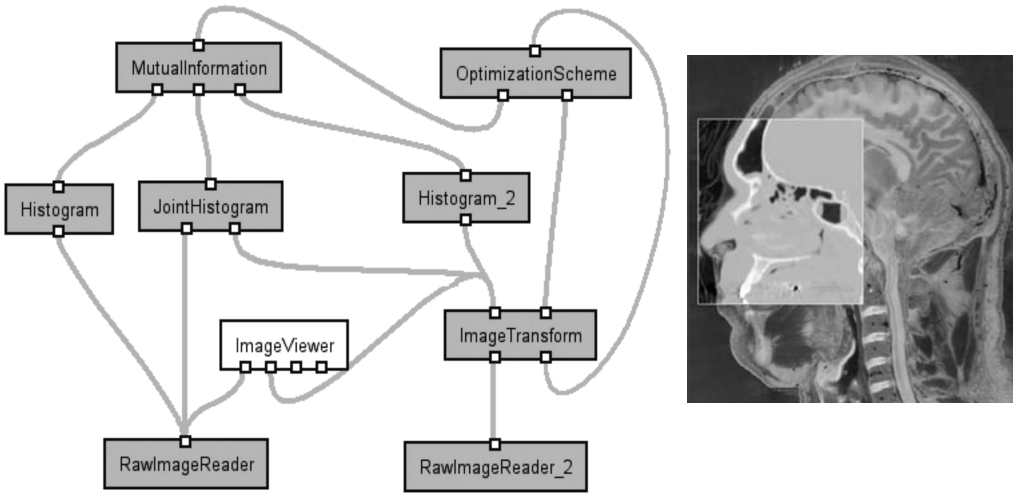


Fig. 6. Volumetric registration pipeline. At first individual histograms of both datasets together with joint histogram are calculated. Powell's optimization routine is used to minimize the mutual information misregistration function for a given set of variable parameters. The whole optimization process is visualized using 2D image viewer.

In the case of volumetric datasets (especially when dealing with different modalities), there is usually no simple relationship between intensities in both datasets. Because of this more sophisticated methods have to be applied to find the optimal alignment. One of the most popular voxel based methods is called *mutual information* [15]. This approach is based on Shannon's information theory to measure the generalized mutual dependence of two variables [14]. In this case misregistration function is defined as follows:

$$C(v) = -\sum_{g_i \in G_O} \sum_{g_j \in G_M} P(g_{i,} g_j) \log \frac{P(g_i, g_j)}{P(g_i)P(g_j)} \qquad (2)$$

where $P(g_i, g_j)$ is a probability distribution of the scatter-plot histogram, and the $P(g_i)$, $P(g_j)$ are the probabilities for each *object* and *model* grey-value in the intersection volume. G_O and G_M are grey-value sets of the *model* and *object* volumes. In general, mutual information is a measure of statistical dependency between two datasets. Unlike in the surface similarity approach, where the similarity assessment takes place in Euclidean space, this match quality measure works in a different mathematical space and estimates the distance between the probability distribution of the scatter-plot histogram $P(g_i, g_j)$ and distribution $P(g_i)P(g_j)$ of two completely independent signals. In case of mutual information function it is difficult to estimate its derivatives. Because of that we may not use gradient guided optimization routines. However there exists a wide range of algorithms (like used in our system Powell's method [13]) where it is sufficient to compute only function values for a given set of parameters. Illustration for the whole process pipeline of volumetric registration is presented in Figure 6.

7. FUZZY SEGMENTATION PIPELINE

Segmentation subdivides an image into objects of interests. These objects are often referred to as regions of interest (ROI) or volumes of interest (VOI). Our segmentation method [2] is capable of segmenting from MR images of the human brain such structures as white matter, grey matter, cerebro-spinal fluid and fat, skin, eyes. Our segmentation algorithm consists of several stages. At first stage, the standard region growing algorithm [4] eliminates the noisy background. This step is done by the RegionGrowingSegmentation Module with a corner points as seed points for region growing algorithm (see Figure 7).

The second step is based on the "fuzzy entropic algorithm" [3] and creates a fuzzy histogram using following equation:

$$H_{fuzzy}(H) = \frac{1}{n \ln 2} \sum_{i=1}^{n} Sn(\mu_H(g_i))h_i \qquad (3)$$

where H is the image histogram with n grey-levels within the fuzzy region g_i and width h_i pixels in the i-th histogram bin. The purpose of that step is to threshold the image histogram to find the local minima that designate the pixel values for tissues in MR scan. This stage is represented by the FuzzyHistogram Module (see Figure 7).

The third stage of the presented algorithm is the pixel classification process. We use a fuzzy inference mechanism with IF-THEN rules.

Example of IF- THEN rules for MRI T1 images:

IF *pixel* IS *bright* AND *Pixel* IS NOT *far* THEN *pixel* IS *white matter*

IF *pixel* IS *grey* AND *Pixel* IS NOT *far* THEN *pixel* IS *grey matter*

IF *pixel* IS *very bright* AND *Pixel* IS NOT *close* THEN *pixel* IS *CSF*, *muscle*, *fat*

IF *pixel* IS *dark* AND *Pixel* IS NOT *close* THEN *pixel* IS *bone*

In the next step we have to find the geometrical centre of processed data set and for each pixel we have to create own trapezoid membership function for a close, an average and a far distance from centre. We have also decided to choose the *G* function [16] as the membership function for the pixel intensity. This membership function is generated automatically on the basis of data obtained from the fuzzy-entropic stage. The inference process is Takagi-Sugeno type [16], where the AND operator is defined as a minimum operator [16]. The defuzzyfication process is executed by the maximum height method [16].

This stage is divided into two phases. The first one is the classification of the brain white and grey matter. Only the first pixel classified as the white matter by the inference mechanism is found iteratively from the geometrical centre. This pixel is a seed for the 3D region growing algorithm with the fuzzy inference as a homogeneity criterion. After the white and grey matter tissues are classified, the second phase classifies all remaining pixels, excluding background. In this phase image pixels are examined sequentially with the fuzzy inference as the decision criterion. In the pixel classification process we have to apply a low pass filter with the 3x3x3 mask to prevent classifying the false single pixel. This stage is performed by the FuzzySegmentation Module (see Figure 7).

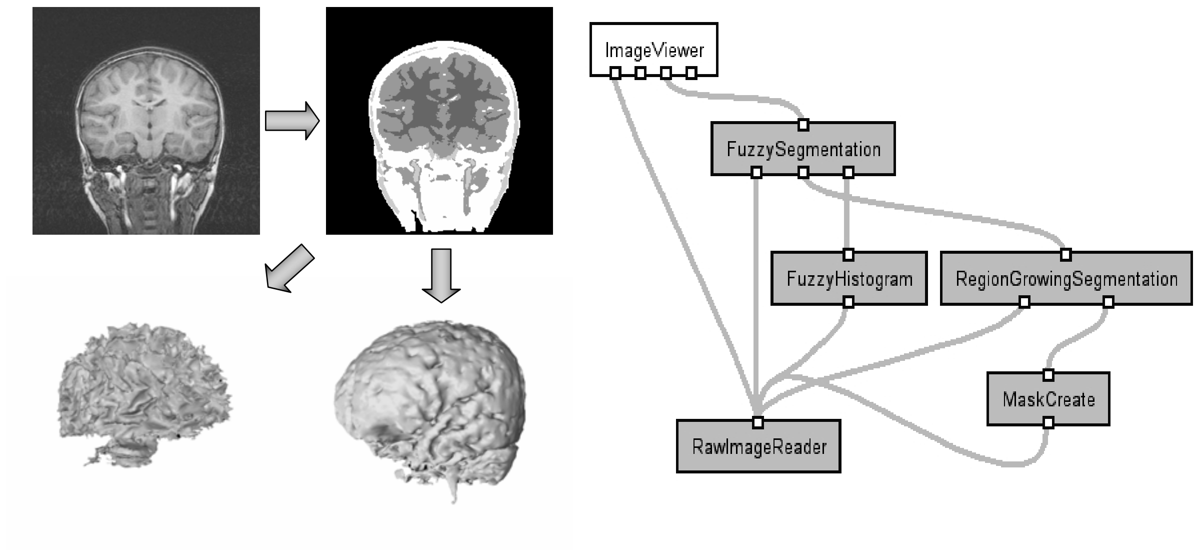


Fig. 7. Fuzzy segmentation pipeline. At first the fuzzy histogram is calculated, and the region growing algorithm removes the noisy background. The main step of the fuzzy segmentation is performed by the FuzzySegmentation Module. Results can be visualized by the ImageViewer Module as 2D cross-sections or as an iso-surface using surface rendering technique.

8. CONCLUSIONS

The *SemiVis framework* and its plug-ins provide a tool for visualization, manipulation and processing of medical datasets of various types. Its extensibility and adaptability allow the user to tailor and modify the systems capabilities to suit his/her own context. Included modules are sufficient for typical medical image processing and add-on tools give the possibility of such sophisticated processing functions as automatic registering or fuzzy segmentation.

BIBLIOGRAPHY

[1] BOOCH G., RUMBAUGH J., JACOBSON I., UML: przewodnik użytkownika, Wydawnictwa Naukowo-Techniczne, Warszawa, 2002.
[2] DENKOWSKI M., CHLEBIEJ M., MIKOŁAJCZAK P., Segmentation of human brain MR image using rule-based fuzzy logic inference, Studies in Health Technology, Kraków, March 2004.
[3] FLEURY M., HAYAT L., CLARK A.F., Parallel entropic auto-thresholding, Image and Vision Computing, vol. 14, April, 1995.
[4] GONZALEZ R.C, WOODS R.E., Digital image processing, Addison-Wesley Publishing Company, Inc., 1992.
[5] HAMLET D., Podstawy techniczne inżynierii oprogramowania, Wydawnictwa Naukowo-Techniczne, Warszawa, 2003.
[6] http://medical.nema.org/
[7] http://www.gnu.org/copyleft/gpl.html
[8] http://www.kitware.com
[9] http://www.trolltech.com
[10] http://www.w3.org/XML/
[11] MARQUARDT D.W.: An Algorithm for Least-Squares Estimation of Nonlinear Parameters, Journal of the Society of Industrial and Applied Mathematics, Vol. 11, pp. 431-331, 1963.
[12] NEIDER J., DAVIS T., WOO M., OpenGL Programming Guide: The official guide to learning OpenGL, Release 1, (Addison Wesley, 1993).

[13] PRESS W.H., FLANNERY B.P., TEUKOLSKY S.A., VETTERLING W.T., Numerical Recipes in C, Cambridge University Press, second edition, 1992.

[14] SHANNON C.E., A mathematical theory of communication, Bell System Technical Journal, Vol. 27, No. 5, pp. 63-70, 1990.

[15] VIOLA P., Alignment by maximization of mutual information, PhD thesis, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, 1995.

[16] ZADEH L.A., Fuzzy Sets and their Applications to Cognitive and Decision Process, pp. 1-39, Academic Press, London, 1975.